Theoretische Informatik
Beweisideen 101

1 Grundbegriffe

Für eine Menge A bezeichnet $|A|$ die Kardinalität von A und $\mathcal{P}(A) = \{S \mid S \subseteq A\}$ die Potenzmenge von A.

In diesem Kurs definieren wir $\mathbb{N} = \{0, 1, 2, \dots\}$.

1.1 Alphabet

Definition Alphabet

Eine endliche, nichtleere Menge Σ heisst **Alphabet**. Die Elemente eines Alphabets werden **Buchstaben (Zeichen, Symbole)** genannt.

Beispiele

- $\Sigma_{\text{bool}} = \{0, 1\}$
- $\Sigma_{\text{lat}} = \{a, ..., z\}$
- $\Sigma_{\text{Tastatur}} = \Sigma_{\text{lat}} \cup \{A, ..., Z, \text{␣}, >, <, (,), ..., !\}$
- $\Sigma_{\text{logic}} = \{0, 1, (,), \wedge, \vee, \neg\}$
- $\Sigma_{abc} = \{a, b, c\}$ (**unser Beispiel für weitere Definitionen**)

1.2 Wort

Definition Wort

- Sei Σ ein Alphabet. Ein **Wort** über Σ ist eine endliche (eventuell leere) Folge von Buchstaben aus Σ.
- Das **leere Wort** λ ist die leere Buchstabenfolge.
- Die **Länge** $|w|$ eines Wortes w ist die Länge des Wortes als Folge, i.e. die Anzahl der Vorkommen von Buchstaben in w.
- Σ^* ist die Menge aller Wörter über Σ. $\Sigma^+ := \Sigma^* \setminus \{\lambda\}$ ist Menge aller nichtleeren Wörter über Σ.
- Seien $x \in \Sigma^*$ und $a \in \Sigma$. Dann ist $|x|_a$ definiert als die Anzahl der Vorkommen von a in x.

Achtung Metavariablen! I.e. Das a in der Definition ist steht für einen beliebigen Buchstaben aus Σ und **nicht** nur für den Buchstaben 'a', der in Σ sein könnte.

Bemerkungen

- Wir schreiben Wörter ohne Komma, i.e. eine Folge $x_1, x_2, ..., x_n$ schreiben wir $x_1x_2...x_n$.
- $|\lambda| = 0$ aber $|␣| = 1$ von Σ_{Tastatur}.
- Der Begriff **Wort** als Fachbegriff der Informatik entspricht **nicht** der Bedeutung des Begriffs Wort in natürlichen Sprachen!
- E.g. Mit ␣ kann der Inhalt eines Buches oder ein Programm als ein Wort über Σ_{Tastatur} betrachtet werden.

Beispiel Verschiedene Wörter über Σ_{abc}:

a, aa, aba, cba, $caaaab$ etc.

Die **Verkettung (Konkatenation)** für ein Alphabet Σ ist eine Abbildung Kon: $\Sigma^* \times \Sigma^* \to \Sigma^*$, so dass

$$\text{Kon}(x, y) = x \cdot y = xy$$

für alle $x, y \in \Sigma^*$.

- Die Verkettung Kon (i.e. Kon von einem Kon (über das gleiche Alphabet Σ)) ist eine assoziative Operation über Σ^*.

$$\text{Kon}(u, \text{Kon}(v, w)) = \text{Kon}(\text{Kon}(u, v), w), \ \forall u, v, w \in \Sigma^*$$

- $x \cdot \lambda = \lambda \cdot x = x, \ \forall x \in \Sigma^*$

- $\implies$ (Σ^*, Kon) ist ein Monoid mit neutralem Element λ.
- Kon nur kommutativ, falls $|\Sigma| = 1$.
- $|xy| = |x \cdot y| = |x| + |y|$. (Wir schreiben ab jetzt xy statt $\text{Kon}(x, y)$)

Beispiel

Wir betrachten wieder Σ_{abc}. Sei $x = abba$, $y = cbcbc$, $z = aaac$.

- $\text{Kon}(x, \text{Kon}(y, z)) = \text{Kon}(x, yz) = xyz = abbacbcbcaaac$
- $|xy| = |abbacbcbc| = 9 = 4 + 5 = |abba| + |cbcbc| = |x| + |y|$

Für eine Wort $a = a_1a_2...a_n$, wobei $\forall i \in \{1, 2, ..., n\}$. $a_i \in \Sigma$, bezeichnet $a^{\text{R}} = a_n a_{n-1}...a_1$ die **Umkehrung (Reversal)** von a.

Sei Σ ein Alphabet. Für alle $x \in \Sigma^*$ und alle $i \in \mathbb{N}$ definieren wir die i-te **Iteration** x^i von x als

$$x^0 = \lambda, x^1 = x \text{ und } x^i = xx^{i-1}.$$

Beispiel

Wir betrachten wieder Σ_{abc}. Sei $x = abba$, $y = cbcbc$, $z = aaac$.

- $z^{\text{R}} = (aaac)^{\text{R}} = caaa$
- $x^{\text{R}} = (abba)^{\text{R}} = abba$
- $x^0 = \lambda$
- $y^2 = yy^{2-1} = yy = cbcbccbcbc$
- $z^3 = zz^2 = zzz = aaacaaacaaac$
- $(x^{\text{R}}z^{\text{R}})^{\text{R}} = ((abba)^{\text{R}}(aaac)^{\text{R}})^{\text{R}} = (abbacaaa)^{\text{R}} = aaacabba$

Seien $v, w \in \Sigma^*$ für ein Alphabet Σ.

- v heisst ein **Teilwort** von $w \iff \exists x, y \in \Sigma^* : \ w = xvy$
- v heisst ein **Präfix** von $w \iff \exists y \in \Sigma^* : \ w = vy$
- v heisst ein **Suffix** von $w \iff \exists x \in \Sigma^* : \ w = xv$
- $v \neq \lambda$ heisst ein **echtes** Teilwort (Präfix, Suffix) von $w \iff v \neq w$ und v Teilwort(Präfix, Suffix) von w

Beispiel

Wir betrachten wieder Σ_{abc}. Sei $x = abba$, $y = cbcbc$, $z = aaac$.

- *bc* ist ein echtes Suffix von y
- *abba* ist kein echtes Teilwort von x.
- *cbcb* ist ein echtes Teilwort und echtes Präfix von y.
- *ac* ist ein echtes Suffix.
- *abba* ist ein Suffix, Präfix und Teilwort von x.

Aufgabe 1

Sei Σ ein Alphabet und sei $w \in \Sigma^*$ ein Wort der Länge $n \in \mathbb{N} \setminus \{0\}$. Wie viele unterschiedliche Teilwörter kann w höchstens haben?

Lösung

Wir haben $w = w_1 w_2 ... w_n$ mit $w_i \in \Sigma$ für $i = 1, ..., n$. Wie viele Teilwörter beginnen mit w_1? Wie viele Teilwörter beginnen mit w_2?

Wir haben also $n + (n-1) + ... + 1 = \frac{n(n+1)}{2}$ Teilwörter. Etwas fehlt aber in unserer Berechnung...

Das leere Wort λ ist auch ein Teilwort! Also haben wir $\frac{n(n+1)}{2} + 1$ Teilwörter.

Aufgabe 2

Sei $\Sigma = \{a, b, c\}$ und $n \in \mathbb{N}$. Bestimme die Anzahl der Wörter aus Σ^n, die das Teilwort a enthalten.

Lösung

In solchen Aufgaben ist es manchmal einfach, das Gegenteil zu berechnen und so auf die Lösung zu kommen. Wie viele Wörter aus Σ^n enthalten das Teilwort a **nicht**?

Da wir jetzt die Anzahl Wörter der Länge n wollen, die nur b und c enthalten, kommen wir auf $|\{b, c\}|^n = 2^n$.

Daraus folgt, dass genau $|\Sigma|^n - 2^n = 3^n - 2^n$ Wörter das Teilwort a enthalten.

Aufgabe 3

Sei $\Sigma = \{a, b, c\}$ und $n \in \mathbb{N} \setminus \{0\}$. Bestimme die Anzahl der Wörter aus Σ^n, die das Teilwort aa nicht enthalten.

Lösung

Wir bezeichnen die Menge aller Wörter mit Länge n über Σ, die aa nicht enthalten als L_n.

Schauen wir mal die ersten zwei Fälle an:

- $L_1 = \{a, b, c\} \implies |L_1| = 3$
- $L_2 = \{ab, ac, ba, bb, bc, ca, cb, cc\} \implies |L_2| = 8$

Nun können wir für $m \geq 3$ jedes Wort $w \in L_m$ als Konkatination $w = x \cdot y \cdot z$ schreiben, wobei wir zwei Fälle unterscheiden:

(a) $\boldsymbol{z \neq a}$

In diesem Fall kann $y \in \{a, b, c\}$ sein, ohne dass die Teilfolge aa entsteht und somit ist xy ein beliebiges Wort aus L_{m-1}.

Dann könnten wir alle Wörter in diesem Case durch $L_{m-1} \cdot \{b, c\}$ beschreiben, was uns die Kardinalität $2 \cdot |L_{m-1}|$ gibt.

(b) $\boldsymbol{z = a}$

In diesem Fall muss $y \neq a$ sein, da sonst aa entstehen würde.

Somit kann xy nur in b oder c enden. x kann aber ein beliebiges Wort der Länge $m-2$ sein.

Deshalb können wir alle Wörter in diesem Case durch $L_{m-2} \cdot \{b, c\} \cdot \{a\}$ beschreiben. Kardinalität: $2 \cdot |L_{m-2}|$.

Sei $\Sigma = \{s_1, s_2, ..., s_m\}, m \geq 1$, ein Alphabet und sei $s_1 < s_2 < ... < s_m$ eine Ordnung auf Σ. Wir definieren die **kanonische Ordnung** auf Σ^* für $u, v \in \Sigma^*$ wie folgt:

$$u < v \iff |u| < |v| \vee (|u| = |v| \wedge u = x \cdot s_i \cdot u' \wedge x \cdot s_j \cdot v')$$
$$\text{für irgendwelche } x, u', v' \in \Sigma^* \text{ und } i < j.$$

Sei $\Sigma_{abc} = \{a, b, c\}$ und wir betrachten folgende Ordnung auf Σ_{abc}: $c < a < b$.

Was wäre die kanonische Ordnung folgender Wörter?

c, abc, $aaac$, $aaab$, $bacc$, a, λ

λ, c, a, abc, $aaac$, $aaab$, $bacc$

1.3 Sprache

Eine **Sprache** L über einem Alphabet Σ ist eine Teilmenge von Σ^*.

- Das Komplement $L^{\complement}$ der Sprache L bezüglich Σ ist die Sprache $\Sigma^* \setminus L$.
- $L_\emptyset = \emptyset$ ist die **leere Sprache**.
- $L_\lambda = \{\lambda\}$ ist die einelementige Sprache, die nur aus dem leeren Wort besteht.

Konkatenation von Sprachen

Sind L_1 und L_2 Sprachen über Σ, so ist

$$L_1 \cdot L_2 = L_1 L_2 = \{vw \mid v \in L_1 \text{ und } w \in L_2\}$$

die **Konkatenation** von L_1 und L_2.

Ist L eine Sprache über Σ, so definieren wir

$$L^0 := L_\lambda \text{ und } L^{i+1} := L^i \cdot L \text{ für alle } i \in \mathbb{N},$$
$$L^* = \bigcup_{i \in \mathbb{N}} L^i \text{ und } L^+ = \bigcup_{i \in \mathbb{N}\setminus\{0\}} L^i = L \cdot L^*.$$

L^* nennt man den **Kleene'schen Stern** von L.

Man bemerke, dass $\Sigma^i = \{x \in \Sigma^* \mid |x| = i\}$, $L_\emptyset L = L_\emptyset = \emptyset$ und $L_\lambda \cdot L = L$.

Mögliche Sprachen über Σ_{abc}

- $L_1 = \emptyset$
- $L_2 = \{\lambda\}$
- $L_3 = \{\lambda, ab, baca\}$
- $L_4 = \Sigma^*_{abc}$, $L_5 = \Sigma^+_{abc}$, $L_6 = \Sigma_{abc}$ oder $L_7 = \Sigma^{27}_{abc}$
- $L_8 = \{c\}^* = \{c^i \mid i \in \mathbb{N}\}$
- $L_9 = \{a^p \mid p \text{ ist prim.}\}$
- $L_{10} = \{c^i a^{3i^2} b a^i c \mid i \in \mathbb{N}\}$

λ ist ein Wort über jedes Alphabet. Aber es muss nicht in jeder Sprache enthalten sein!

Seien L_1, L_2 und L_3 Sprachen über einem Alphabet Σ. Dann gilt

$$L_1L_2 \cup L_1L_3 = L_1(L_2 \cup L_3) \tag{1}$$
$$L_1(L_2 \cap L_3) \subseteq L_1L_2 \cap L_1L_3 \tag{2}$$

Weshalb nicht '=' bei (2)?

Sei $\Sigma = \Sigma_{\text{bool}} = \{0, 1\}$, $L_1 = \{\lambda, 1\}$, $L_2 = \{0\}$ und $L_3 = \{10\}$.

Dann haben wir $L_1(L_2 \cap L_3) = \emptyset \neq \{10\} = L_1L_2 \cap L_1L_3$.

Beweise im Buch/Vorlesung

Seien Σ_1 und Σ_2 zwei beliebige Alphabete. Ein Homomorphismus von Σ_1^* nach Σ_2^* ist jede Funktion $h : \Sigma_1^* \to \Sigma_2^*$ mit den folgenden Eigenschaften:

(i) $h(\lambda) = \lambda$ und

(ii) $h(uv) = h(u) \cdot h(v)$ für alle $u, v \in \Sigma_1^*$.

Wir können Probleme etc. in anderen Alphabeten kodieren. So wie wir verschiedenste Konzepte, die wir auf Computer übertragen in Σ_{bool} kodieren.

2 Algorithmische Probleme

Mathematische Definition folgt in Kapitel 4 (Turingmaschinen).

Vorerst betrachten wir Programme, die **für jede zulässige Eingabe halten und eine Ausgabe liefern**, als Algorithmen.
Wir betrachten ein Programm (Algorithmus) A als Abbildung $A : \Sigma_1^* \to \Sigma_2^*$ für beliebige Alphabete Σ_1 und Σ_2. Dies bedeutet, dass

(i) die Eingaben als Wörter über Σ_1 kodiert sind,

(ii) die Ausgaben als Wörter über Σ_2 kodiert sind und

(iii) A für jede Eingabe eine eindeutige Ausgabe bestimmt.

A und B äquivalent $\iff$ Eingabealphabet Σ gleich, $A(x) = B(x), \forall x \in \Sigma^*$

Ie. diese Notion von "Äquivalenz" bezieht sich nur auf die Ein und Ausgabe.

Das **Entscheidungsproblem** (Σ, L) für ein gegebenes Alphabet Σ und eine gegebene Sprache $L \subseteq \Sigma^*$ ist, für jedes $x \in \Sigma^*$ zu entscheiden, ob

$$x \in L \text{ oder } x \notin L.$$

Ein Algorithmus A **löst** das Entscheidungsproblem (Σ, L), falls für alle $x \in \Sigma^*$ gilt:

$$A(x) = \begin{cases} 1, & \text{falls } x \in L, \\ 0, & \text{falls } x \notin L. \end{cases}$$

Wir sagen auch, dass A die Sprache L erkennt.

Wenn für eine Sprache L ein Algorithmus existiert, der L erkennt, sagen wir, dass L **rekursiv** ist.

Wir sind oft an spezifischen Eigenschaften von Wörtern aus Σ^* interessiert, die wir mit einer Sprache $L \subseteq \Sigma^*$ beschreiben können.

Dabei sind dann L die Wörter, die die Eigenschaft haben und $L^\complement = \Sigma^* \setminus L$ die Wörter, die diese Eigenschaft nicht haben.

Jetzt ist die allgemeine Formulierung von Vorteil!

i. **Primzahlen finden**:

   Entscheidungsproblem $(\Sigma_{\text{bool}}, L_p)$ wobei
   $L_p = \{x \in (\Sigma_{\text{bool}})^* \mid \text{Nummer}(x) \text{ ist prim}\}$.

ii. **Syntaktisch korrekte Programme**:

Entscheidungsproblem $(\Sigma_{\text{Tastatur}}, L_{C++})$ wobei
$L_{C++} = \{x \in (\Sigma_{\text{Tastatur}})^* \mid x \text{ ist ein syntaktisch korrektes C++ Programm}\}$.

iii. **Hamiltonkreise finden:**

Entscheidungsproblem (Σ, HK) wobei $\Sigma = \{0, 1, \#\}$ und
HK$= \{x \in \Sigma^* \mid x \text{ kodiert einen Graphen, der einen Hamiltonkreis enthält.}\}$

Äquivalenzprobleme $\subset$ Entscheidungsprobleme

Seien Σ und Γ zwei Alphabete.

- Wir sagen, dass ein Algorithmus A eine **Funktion (Transformation)** $f : \Sigma^* \to \Gamma^*$ **berechnet (realisiert)**, falls
$$A(x) = f(x) \text{ für alle } x \in \Sigma^*$$
- Sei $R \subseteq \Sigma^* \times \Gamma^*$ eine Relation in Σ^* und Γ^*. Ein Algorithmus A **berechnet** R (bzw. **löst das Relationsproblem** R), falls für jedes $x \in \Sigma^*$, für das ein $y \in \Gamma^*$ mit $(x, y) \in R$ existiert, gilt:
$$(x, A(x)) \in R$$

Ein **Optimierungsproblem** ist ein 6-Tupel $\mathcal{U} = (\Sigma_I, \Sigma_O, L, M, \text{cost}, \text{goal})$, wobei:

(i) Σ_I ist ein Alphabet (genannt **Eingabealphabet**),

(ii) Σ_O ist ein Alphabet (genannt **Ausgabealphabet**),

(iii) $L \subseteq \Sigma_I^*$ ist die Sprache der **zulässigen Eingaben** (als Eingaben kommen nur Wörter in Frage, die eine sinnvolle Bedeutung haben). Ein $x \in L$ wird ein **Problemfall (Instanz) von** $\mathcal{U}$ genannt.

(iv) M ist eine Funktion von L nach $\mathcal{P}(\Sigma_O^*)$, und für jedes $x \in L$ ist $M(x)$ die **Menge der zulässigen Lösungen für** x,

(v) **cost** ist eine Funktion, **cost**: $\bigcup_{x \in L}(\mathcal{M}(x) \times \{x\}) \to \mathbb{R}^+$, genannt **Kostenfunktion**,

(vi) **goal** $\in \{\text{Minimum}, \text{Maximum}\}$ ist das **Optimierungsziel**.

Eine zulässige Lösung $\alpha \in \mathcal{M}(x)$ heisst **optimal** für den Problemfall x des Optimierungsproblems $\mathcal{U}$, falls

$$\text{cost}(\alpha, x) = \mathbf{Opt}_{\mathcal{U}}(x) = \text{ goal}\{\text{cost}(\beta, x) \mid \beta \mathcal{M}(x)\}.$$

Ein Algorithmus A **löst** $\mathcal{U}$, falls für jedes $x \in L$

(i) $A(x) \in \mathcal{M}(x)$

(ii) $\text{cost}(A(x), x) = \text{ goal}\{\text{cost}(\beta, x) \mid \beta \in \mathcal{M}(x)\}$.

3 Kolmogorov Komplexität

Sei Σ ein Alphabet und $x \in \Sigma^*$. Wir sagen, dass ein Algorithmus A das Wort x **generiert**, falls A für die Eingabe λ die Ausgabe x liefert.

Beispiel:

```
A_n:    begin
            for i = 1 to n;
                write(01);
        end
```

A_n generiert $(01)^n$.

Sei Σ ein Alphabet und sei $L \subseteq \Sigma^*$. A ist ein **Aufzählungsalgorithmus für** L, falls A für jede Eingabe $n \in \mathbb{N} \setminus \{0\}$ die Wortfolge $x_1, ..., x_n$ ausgibt, wobei $x_1, ..., x_n$ die kanonisch n ersten Wörter in L sind.

Kolmogorov-Komplexität

Für jedes Wort $x \in (\Sigma_{\text{bool}})^*$ ist die **Kolmogorov-Komplexität** $K(x)$ **des Wortes** x das Minimum der binären Längen, der Pascal-Programme, die x generieren.

$K(x)$ ist die kürzestmögliche Länge einer Beschreibung von x.

Die einfachste (und triviale) Beschreibung von x, ist wenn man x direkt angibt.

x kann aber eine Struktur oder Regelmässigkeit haben, die eine Komprimierung erlaubt.

Beispiel

Sei $w = 0101010101010101010101010101010101010101$. Die Länge von w ist $|w| = 40$ und die triviale Beschreibunglänge wäre wie gegeben 40.

Aber durch die Regelmässigkeit von einer 20-fachen Wiederholung der Sequenz 01, können w auch durch $(01)^{20}$ beschreiben. Hierbei ist die Beschreibungslänge ein wenig mehr als 4 Zeichen.

Es existiert eine Konstante d, so dass für jedes $x \in (\Sigma_{\text{bool}})^*$

$$K(x) \leq |x| + d$$

Die **Kolmogorov-Komplexität einer natürlichen Zahl** n ist $K(n) = K(\text{Bin}(n))$.

Für jede Zahl $n \in \mathbb{N} \setminus \{0\}$ existiert ein Wort $w_n \in (\Sigma_{\text{bool}})^n$, so dass

$$K(w_n) \geq |w_n| = n$$

Kapitel 2

Lemma 2.5

Für jede Zahl $n \in \mathbb{N}$ existiert ein Wort $w_n \in (\Sigma_{bool})^n$, so dass

$$K(w_n) \geq |w_n| = n$$

d.h., es existiert für jede Zahl n ein nichtkomprimierbares Wort der Länge n.

Beweis:

Es gibt 2^n Wörter $x_1, ..., x_{2^n}$ über Σ_{bool} der Länge n. Wir bezeichnen $C(x_i)$ als den Bitstring des kürzesten Programms, der x_i generieren kann. Es ist klar, dass für $i \neq j : C(x_i) \neq C(x_j)$.

Die Anzahl der Bitstrings, i.e. der Wörter der Länge $< n$ über Σ_{bool} ist:

$$\sum_{i=1}^{n-1} 2^i = 2^n - 2 < 2^n$$

Also muss es unter den Wörtern $x_1, ..., x_{2^n}$ mindestens ein Wort x_k mit $K(x_k) \geq n$ geben.

■

Satz 2.2

Sei L eine Sprache über Σ_{bool}. Sei, für jedes $n \in \mathbb{N} \setminus \{0\}$, z_n das n-te Wort in L bezüglich der kanonischen Ordnung. Wenn ein Programm A_L existiert, dass das Entscheidungsproblem (Σ_{bool}, L) löst, dann gilt für alle $n \in \mathbb{N} \setminus \{0\}$, dass

$$K(z_n) \leq \lceil \log_2(n+1) \rceil + c$$

wobei c eine von n unabhängige Konstante ist.

Beweisidee:

Wir können aus A_L, ein Programm entwerfen, dass das kanonisch n-te Wort generiert, indem wir in der kanonischen Reihenfolge alle Wörter $x \in (\Sigma_{bool})^*$ durchgehen und mit A_L entscheiden, ob $x \in L$. Dann können wir einen Counter c haben und den Prozess abbrechen, wenn der Counter $c = n$ wird und dann dieses Wort ausgeben.

Wir sehen, dass dieses Programm ausser der Eingabe n immer gleich ist. Sei die Länge dieses Programms c, dann können wir für das n-te Wort der Sprache L, z_n, die Kolmogorov-Komplexität auf n reduzieren, bzw:

$$K(z_n) \leq \lceil \log_2(n+1) \rceil + c$$

■

Lemma 2.6

Sei $n_1, n_2, n_3, ...$ eine steigende unendliche Folge natürlicher Zahlen mit $K(n_i) \geq \lceil \log_2 n_i \rceil / 2$. Für jedes $i \in \mathbb{N} \setminus \{0\}$ sei q_i die grösste Primzahl, die die Zahl n_i teilt. Dann ist die Menge $Q = \{q_i \mid i \in \mathbb{N} \setminus \{0\}\}$ unendlich.

Beweis: Wir beweisen diese Aussage per Widerspruch:

Nehmen wir zum Widerspruch an, dass die Menge $Q = \{q_i \mid i \in \mathbb{N} \setminus \{0\}\}$ sei endlich. Sei q_m die grösste Primzahl in Q. Dann können wir jede Zahl n_i eindeutig als

$$n_i = q_1^{r_{i,1}} \cdot q_2^{r_{i,2}} \cdot \dots \cdot q_m^{r_{i,m}}$$

für irgendwelche $r_{i,1}, r_{i,2}, ..., r_{i,m} \in \mathbb{N}$ darstellen. Sei c die binäre Länge eines Programms, dass diese $r_{i,j}$ als Eingaben nimmt und n_i erzeugt (A ist für alle $i \in \mathbb{N}$ bis auf die Eingaben $r_{i,1}, ..., r_{i,m}$ gleich).

Dann gilt:

$$K(n_i) \leq c + 8 \cdot (\lceil \log_2(r_{i,1}+1) \rceil + \lceil \log_2(r_{i,2}+1) \rceil + ... + \lceil \log_2(r_{i,m}+1) \rceil)$$

Die multiplikative Konstante 8 kommt daher, dass wir für die Zahlen $r_{i,1}, r_{i,2}, ..., r_{i,m}$ dieselbe Kodierung, wie für den Rest des Programmes verwenden (z.B. ASCII-Kodierung), damit ihre Darstellungen eindeutig voneinander getrennt werden können. Weil $r_{i,j} \leq \log_2 n_i, \forall j \in \{1, ..., m\}$ erhalten wir

$$K(n_i) \leq c + 8m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil, \forall i \in \mathbb{N} \setminus \{0\}$$

Weil m und c Konstanten unabhängig von i sind, kann

$$\lceil \log_2 n_i \rceil / 2 \leq K(n_i) \leq c + 8m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil$$

$$\lceil \log_2 n_i \rceil / 2 \leq c + 8m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil$$

nur für endlich viele $i \in \mathbb{N} \setminus \{0\}$ gelten.

Dies ist ein Widerspruch!

Folglich ist die Menge Q unendlich.

■

Kapitel 3

Lemma 3.3

Sei $A = (Q, \Sigma, \delta_A, q_0, F)$ ein EA. Seien $x, y \in \Sigma^*, x \neq y$, so dass

$$\hat{\delta}_A(q_0, x) = p = \hat{\delta}_A(q_0, y)$$

für ein $p \in Q$ (also $x, y \in \mathrm{Kl}[p]$). Dann existiert für jedes $z \in \Sigma^*$ ein $r \in Q$, so dass xz und $yz \in \mathrm{Kl}[r]$, also gilt insbesondere

$$xz \in L(A) \iff yz \in L(A)$$

Beweis:

Aus der Existenz der Berechnungen
$(q_0, x) \vdash_A^* (p, \lambda)$ und $(q_0, y) \vdash_A^* (p, \lambda)$ von A folgt die Existenz der Berechnungen auf xz und yz:
$(q_0, xz) \vdash_A^* (p, z)$ und $(q_0, yz) \vdash_A^* (p, z)$ für alle $z \in \Sigma^*$.
Wenn $r = \hat{\delta}_A(p, z)$ ist, dann ist die Berechnung von A auf xz und yz:
$(q_0, xz) \vdash_A^* (p, z) \vdash_A^* (r, \lambda)$ und $(q_0, yz) \vdash_A^* (p, z) \vdash_A^* (r, \lambda)$.
Wenn $r \in F$, dann sind beide Wörter xz und yz in $L(A)$. Falls $r \notin F$, dann sind $xz, yz \notin L(A)$.

■

Lemma 3.4: Pumping-Lemma

Sei L regulär. Dann existiert eine Konstante $n_0 \in \mathbb{N}$, so dass sich jedes Wort $w \in \Sigma^*$ mit $|w| \geq n_0$ in drei Teile y, x und z zerlegen lässt, das heisst $w = yxz$, wobei

(i) $|yx| \leq n_0$,

(ii) $|x| \geq 1$ und

(iii) entweder $\{yx^kz \mid k \in \mathbb{N}\} \subseteq L$ oder $\{yx^kz \mid k \in \mathbb{N}\} \cap L = \emptyset$.

Beweis:

Sei $L \in \Sigma^*$ regulär. Dann existiert ein EA $A = (Q, \Sigma, \delta_A, q_0, F)$, so dass $L(A) = L$.
Sei $n_0 = |Q|$ und $w \in \Sigma^*$ mit $|w| \geq n_0$. Dann ist $w = w_1w_2...w_{n_0}u$, wobei $w_i \in \Sigma$ für $i = 1, ..., n_0$ und $u \in \Sigma^*$. Betrachten wir die Berechnung auf $w_1w_2...w_{n_0}$:

$$(q_0, w_1w_2w_3...w_{n_0}) \vdash_A (q_1, w_2w_3...w_{n_0}) \vdash_A (q_2, w_3...w_{n_0}) \vdash_A ... \vdash_A (q_{n_0-1}, w_{n_0}) \vdash_A (q_{n_0}, \lambda)$$

In dieser Berechnung kommen n_0+1 Zustände $q_0, q_1, ..., q_{n_0}$ vor. Da $|Q| = n_0$, existieren $i, j \in \{0, 1, ..., n_0\}, i < j$, so dass $q_i = q_j$. Daher haben wir in der Berechnung die Konfigurationen

$$(q_0, w_1w_2w_3...w_{n_0}) \vdash_A^* (q_i, w_{i+1}w_{i+2}...w_{n_0}) \vdash_A^* (q_i, w_{j+1}...w_{n_0}) \vdash_A^* (q_{n_0}, \lambda)$$

Dies impliziert

$$(q_i, w_{i+1}w_{i+2}...w_j) \vdash_A^* (q_i, \lambda) \qquad (1)$$

Wir setzen nun $y = w_1...w_i$, $x = w_{i+1}...w_j$ und $z = w_{j+1}...w_{n_0}u$, so dass $w = yxz$.

Wir überprüfen nun die Eigenschaften (i),(ii) und (iii):

(i) $yx = w_1...w_iw_{i+1}...w_j$ und daher $|yx| = j \leq n_0$.

(ii) Da $|x| \geq j - i$ und $i < j$, ist $|x| \geq 1$.

(iii) (1) impliziert $(q_i, x^k) \vdash_A^* (q_i, \lambda)$ für alle $k \in \mathbb{N}$. Folglich gilt für alle $k \in \mathbb{N}$:

$$(q_0, yx^kz) \vdash_A^* (q_i, x^kz) \vdash_A^* (q_i, z) \vdash_A^* (\hat{\delta}_A(q_i, z), \lambda)$$

Wir sehen, dass für alle $k \in \mathbb{N}$ die Berechnungen im gleichen Zustand $q_{end} = \hat{\delta}_A(q_i, z)$ enden. Falls also $q_{end} \in F$, akzeptiert A alle Wörter aus $\{yx^kz \mid k \in \mathbb{N}\}$. Falls $q_{end} \notin F$, dann akzeptiert A kein Wort aus $\{yx^kz \mid k \in \mathbb{N}\}$.

■

Lemma 3.6

Sei $L_k = \{x1y \mid x \in (\Sigma_{bool})^*, y \in (\Sigma_{bool})^k\}$.

Für alle $k \in \mathbb{N} \setminus \{0\}$ muss jeder EA, der L_k akzeptiert, mindestens 2^k Zustände haben.

Beweis:

Sei $B_k = (Q_k, \Sigma_{bool}, \delta_k, q_{0k}, F_k)$ ein EA mit $L(B_k) = L_k$.

Nach **Lemma 3.3** gilt für $x, y \in (\Sigma_{bool})^*$:

Wenn $\hat{\delta}_k(q_{0k}, x) = \hat{\delta}_k(q_{0k}, y)$, dann gilt für alle $z \in (\Sigma_{bool})^*$:

$$xz \in L(B_k) \iff yz \in L(B_k)$$

Die Idee des Beweises ist es, eine Menge S_k von Wörtern zu finden, so dass für keine zwei unterschiedlichen Wörter $x, y \in S_k$ die Gleichung $\hat{\delta}_k(q_{0k}, x) = \hat{\delta}_k(q_{0k}, y)$ gelten darf. Dann müsste B_k mindestens $|S_k|$ viele Zustände haben.

Wir wählen $S_k = (\Sigma_{bool})^k$ und zeigen, dass $\hat{\delta}_k(q_{0k}, q)$ paarweise unterschiedliche Zustände für alle $u \in S_k$ sind.

Wir beweisen dies per Widerspruch.

Seien $x = x_1x_2...x_k$ und $y = y_1y_2...y_k$ für $x_i, y_i \in \Sigma_{bool}, i \in \{1, ..., k\}$ zwei unterschiedliche Wörter aus S_k.

Nehmen wir zum Widerspruch an, dass $\hat{\delta}_k(q_{0k}, x) = \hat{\delta}_k(q_{0k}, y)$.

Weil $x \neq y$, existiert ein $j \in \{1, ..., k\}$, so dass $x_j \neq y_j$. O.B.d.A. setzen wir $x_j = 1$ und $y_j = 0$. Betrachten wir nun $z = 0^{j-1}$. Dann ist

$xz = x_1...x_{j-1}1x_{j+1}...x_k0^{j-1}$ und $yz = y_1...y_{j-1}0y_{j+1}...y_k0^{j-1}$

und daher $xz \in L_k$ und $yz \notin L_k$. Dies ist ein Widerspruch! Folglich gilt $\hat{\delta}_k(q_{0k}, x) \neq \hat{\delta}_k(q_{0k}, y)$ für alle paarweise unterschiedliche $x, y \in S_k = (\Sigma_{bool})^k$.

Daher hat B_k mindestens $|S_k| = 2^k$ viele Zustände.

■

Kapitel 4

Lemma 4.2

Für jede Mehrband-TM A existiert eine zu A äquivalente TM B.

Beweis:

Sei A eine k-Band-Turingmaschine für ein $k \in \mathbb{N} \setminus \{0\}$. Wir konstruieren eine TM B, die Schritt für Schritt A simuliert.

B speichert die Inhalte aller $k + 1$ Bänder von A auf ihrem einzigen Band. Anschaulich gesprochen ist jedes Feld auf dem Band von B ein $2(k + 1)$-Tupel und jedes Element dieses Tupels ist auf einer Spur. Sei Γ_A das Arbeitsalphabet von A. Dann gilt

$$\Gamma_B = (\Sigma_A \cup \{¢, \$, \sqcup\}) \times \{\sqcup, \uparrow\} \times (\Gamma_A \times \{\sqcup, \uparrow\})^k \cup \Sigma_A \cup \{\sqcup, ¢\}$$

Für ein Symbol $\alpha = (a_0, a_1, a_2, ..., a_{2k+1}) \in \Gamma_B$ sagen wir, dass a_i auf der i-ten Spur liegt. Daher bestimmen die i-ten Elemente der Symbole auf dem Band von B den Inhalt der i-ten Spur. Eine Konfiguration $(q, w, i, x_1, i_1, x_2, i_2, ..., x_k, i_k)$ von A ist dann in B wie folgt gespeichert.

- Der Zustand q ist in der endlichen Kontrolle von B gespeichert.
- Die 0-te Spur des Bandes von B enthält die $¢w\$$ (i.e. den Inhalt des Eingabebandes von A)
- Für alle $i \in \{1, ..., k\}$ enthält die $(2i)$-te Spur des Bandes von B den Inhalt vom i-ten Band von A (i.e. $¢x_i\$$).
- Für alle $i \in \{1, ..., k\}$ bestimmt die $(2i + 1)$-te Spur des Bandes von B mit dem Symbol $\uparrow$ die Position des Kopfes auf dem i-ten Arbeitsband von A.

Ein Schritt von A kann jetzt durch folgende Prozedur von B simuliert werden:

1. B liest einmal den Inhalt ihres Bandes von links nach rechts, bis sie alle $k+1$ Kopfpositionen von A gefunden hat, und speichert dabei in ihrem Zustand die $k+1$ Symbole, die an diesen Positionen stehen. (Dies kann ohne weiteres in der Zustandsmenge abgespeichert werden, da k fix ist, folglich ist dann Γ_A^k auch endlich)
2. Nach der ersten Phase kennt B das ganze Argument (der Zustand von A ist im Zustand von B gespeichert) der Transitionsfunktion von A und kann also die entsprechenden Aktionen (Köpfe bewegen, Ersetzen von Symbolen) von A bestimmen. Diese Änderungen führt B in einem Lauf über ihr Band von rechts nach links durch.

■

Kapitel 5

Satz 5.4

$\mathcal{P}((\Sigma_{bool})^*)$ ist nicht abzählbar.

Beweis:

Wir definieren eine injektive Funktion von $f : [0,1] \to \mathcal{P}((\Sigma_{bool})^*)$ und beweisen so $|\mathcal{P}((\Sigma_{bool})^*)| \geq |[0,1]|$.

Sei $a \in [0,1]$ beliebig. Wir können a wie folgt binär darstellen: Nummer$(a) = 0.a_1a_2a_3a_4...$ mit $a = \sum_{i=1}^{\infty} a_i \cdot 2^{-i}$. Hier ist zu beachten, dass wir für eine Zahl a immer die lexikographisch letzte Darstellung. Dies tun wir, weil eine reelle Zahl 2 verschiedene Binärdarstellungen haben kann. Beispiel: $\frac{1}{2} = 0.1\overline{0} = 0.0\overline{1}$.

Für jedes a definieren wir:

$$f(a) = \{a_1, a_2a_3, a_4a_5a_6, ..., a_{\binom{n}{2}+1}a_{\binom{n}{2}+2}...a_{\binom{n+1}{2}}, ...\}$$

Da $f(a) \subseteq (\Sigma_{bool})^*$ gilt $f(a) \in \mathcal{P}((\Sigma_{bool})^*)$.

Wir haben für alle $n \in \mathbb{N} \setminus \{0\}$, dass $f(a)$ **genau** ein Wort dieser Länge enthält. Nun können wir daraus folgendes schliessen:

Weil die Binärdarstellung zweier unterschiedlichen reellen Zahlen an mindestens einer Stelle unterschiedlich ist, gilt $b \neq c \implies f(b) \neq f(c), \forall b, c \in [0,1]$.

Folglich ist f injektiv und wir haben $|\mathcal{P}((\Sigma_{bool})^*)| \geq |[0,1]|$.

Da $[0,1]$ nicht abzählbar ist, folgt daraus:

$\mathcal{P}((\Sigma_{bool})^*)$ ist nicht abzählbar.

■

Satz 5.5

$L_{\text{diag}} \notin \mathcal{L}_{\text{RE}}$.

Beweis:

Wir haben

$$L_{\text{diag}} = \{w \mid w = w_i \text{ und } M_i \text{ akzeptiert } w_i \text{ nicht für ein } i \in \mathbb{N} \setminus \{0\}\}$$

Widerspruchsbeweis:

Sei $L_{\text{diag}} \in \mathcal{L}_{\text{RE}}$. Dann existiert eine TM M, so dass $L(M) = L_{\text{diag}}$. Da diese TM eine TM in der Nummerierung aller TM ist, existiert ein $i \in \mathbb{N}$, so dass $M_i = M$.

Wir betrachten nun das Wort w_i für diese $i \in \mathbb{N}$. Per Definition von L_{diag}, gilt:

$$w_i \in L_{\text{diag}} \iff w_i \notin L(M_i)$$

Da aber $L(M_i) = L_{\text{diag}}$, haben wir folgenden Widerspruch:

$$w_i \in L_{\text{diag}} \iff w_i \notin L_{\text{diag}}$$

Folglich gilt $L_{\text{diag}} \notin \mathcal{L}_{\text{RE}}$.

■

Lemma 5.4

Sei Σ ein Alphabet. Für jede Sprache $L \subseteq \Sigma^*$ gilt:

$$L \leq_{\text{R}} L^{\complement} \text{ und } L^{\complement} \leq_{\text{R}} L$$

Beweis: Es reicht $L^{\complement} \leq_{\text{R}} L$ zu zeigen, da $(L^{\complement})^{\complement} = L$ und somit dann $(L^{\complement})^{\complement} = L \leq_{\text{R}} L^{\complement}$.

Sei M' ein Algorithmus für L, der immer hält ($L \in \mathcal{L}_{\text{R}}$). Dann beschreiben wir einen Algorithmus B, der $L^{\complement}$ entscheidet.

B übernimmt die Eingaben und gibt sie an M' weiter und invertiert dann die Entscheidung von M'. Weil M' immer hält, hält auch B immer und wir haben offensichtlich $L(B) = L$.

■

Korollar 5.2 (bzw. Anwendung von Lemma 5.4)

$(L_{\text{diag}})^{\complement} \notin \mathcal{L}_{\text{R}}$.

Beweis:

Aus Lemma 5.4 haben wir $L_{\text{diag}} \leq_{\text{R}} (L_{\text{diag}})^{\complement}$. Daraus folgt $L_{\text{diag}} \notin \mathcal{L}_{\text{R}} \implies (L_{\text{diag}})^{\complement} \notin \mathcal{L}_{\text{R}}$. Da $L_{\text{diag}} \notin \mathcal{L}_{\text{RE}}$ gilt auch $L_{\text{diag}} \notin \mathcal{L}_{\text{R}}$.

Folglich gilt $(L_{\text{diag}})^{\complement} \notin \mathcal{L}_{\text{R}}$.

■

Lemma 5.8

$L_{\text{H},\lambda} \notin \mathcal{L}_{\text{R}}$.

Beweis:

Wir zeigen $L_{\text{H}} \leq_{\text{EE}} L_{\text{H},\lambda}$. Wir beschreiben einen Algorithmus B, so dass $x \in L_{\text{H}} \iff B(x) \in L_{\text{H},\lambda}$.

Für jede Eingabe arbeitet B wie folgt:

- Falls x von der falschen Form, dann $B(x) = M_{inf}$, wobei M_{inf} unabhängig von der Eingabe immer unendlich läuft.
- Sonst $x = \text{Kod}(M)\#w$: Dann $B(x) = M'$, wobei M' die Eingabe ignoriert und immer M auf w simuliert.

Wir sehen, dass M' genau dann auf λ hält, wenn $x \in L_{\text{H}}$.

Daraus folgt $x \in L_{\text{H}} \iff B(x) \in L_{\text{H},\lambda}$.

■

Kapitel 6

Lemma 6.1

Sei k eine positive ganze Zahl. Für jede k-Band Turingmaschine A, die immer hält, existiert eine äquivalente 1-Band-TM B, so dass

$$\text{Space}_B(n) \leq \text{Space}_A(n)$$

Beweisskizze:

Gleiche Konstruktion wie in Lemma 4.2. Wir können leicht sehen, dass B genau so viele Felder braucht, wie A.

Lemma 6.2

Zu jeder MTM A existiert eine äquivalente MTM B mit

$$\text{Space}_B(n) \leq \frac{\text{Space}_A(n)}{2} + 2$$

Beweisskizze:

Wir fassen jeweils 2 Felder von A zu einem Feld in B zusammen. $\Gamma_B = \Gamma_A \times \Gamma_A$. Wir addieren 1 für das ¢ am linken Rand und 1 für das Aufrunden im Fall von ungerader Länge.

Lemma 6.3

$\text{TIME}(t) \subseteq \text{SPACE}(t)$

Beweisskizze: In t Schritten sind höchstens t Felder beschreibbar.

Lemma 6.4

Sei S platzkonstruierbar. Für jede MTM M, für welche $\text{Space}_M(w) \leq s(|w|)$ nur für alle $w \in L(M)$ erfüllt, existiert eine äquivalente MTM M', welche dies für alle $w \in \Sigma^*$ erfüllt.

Beweisskizze: Erzeuge für jede Eingabe $x \in \Sigma^*$ zuerst $0^{s(|x|)}$ auf einem zusätzlichen Band und nutze das als Platzüberwachung. Wenn M' diesen Platz überschreiten will, wird die Simulation unterbrochen und die Eingabe verworfen.

Lemma 6.5

Sei t zeitkonstruierbar. Zu jeder MTM, welche $\text{Time}_M(w) \leq t(|w|)$ nur für alle $w \in L(M)$ erfüllt, existiert eine äquivalente MTM M', welche zumindest $\text{Time}_M(w) \leq 2t(|w|) \in \mathcal{O}(t(|w|))$ für alle $w \in \Sigma^*$ erfüllt.

Beweisskizze: Schreibe für jede Eingabe $x \in \Sigma^*$ $0^{t(|x|)}$ auf ein zusätzliches Arbeitsband und nutze dies zur Zeitzählung. Wenn M' mehr Schritte machen will, wird die Simulation abgebrochen und die Eingabe verworfen.

Satz 6.2

Für jede Funktion s mit $s(n) \geq \log_2(n)$ gilt:

$$\textbf{SPACE}(s(n)) \subseteq \bigcup_{c \in \mathbb{N}} \textbf{TIME}(c^{s(n)})$$

Beweis:

Sei $L \in \mathbf{SPACE}(s(n))$. Nach Lemma 6.1 existiert eine 1-Band-TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, die **immer hält**, so dass $L = L(M)$ und $\text{Space}_M(n) \leq d \cdot s(n)$ für $d \in \mathbb{N}$ gelten. Für jede Konfiguration $C = (q, w, i, x, j)$ von M definieren wir die **innere Konfiguration von** C als

$$\text{In}(C) = (q, i, x, j).$$

Die innere Konfiguration enthält das Eingabewort w nicht, da dies sich während einer Berechnung nicht ändert.

Wir betrachten die Menge aller inneren Konfigurationen , dass bei einer **deterministischen** TM jede Berechnung $D = C_1, C_2, C_3, ...$ von M auf einem Wort w mit $|w| = n$, die länger als

4 EE-Reduktionen und R-Reduktionen – Komplexitätsbeweise

Mit Inspiration von der Zsf. von Fabian Frei

Generelle Bemerkungen:

- L rekursiv (entscheidbar) $\iff$ $L \in \mathcal{L}_{\text{R}}$
- L rekursiv aufzählbar $\iff$ $L \in \mathcal{L}_{\text{RE}}$
- "Algorithmus" ist ein anderes Wort für eine Turingmaschine, die **immer** terminiert.

4.1 $L \in \mathcal{L}_{\mathbf{R}}$

Wir kennen zwei Methoden um dies zu beweisen:

- Wir finden eine Sprache $L' \in \mathcal{L}_{\text{R}}$ und zeigen $L \leq_{\text{R}} L'$. (Meistens ein wenig umständlich)
- Direkter Beweis: Eine TM (bzw. ein Algorithmus) A beschreiben, so dass $L(A) = L$ und A immer terminiert.

4.2 $L \notin \mathcal{L}_{\mathbf{R}}$

Wir kennen hier auch 3 Arten:

- Folgt sofort aus $L \notin \mathcal{L}_{\text{RE}}$, da $\mathcal{L}_{\text{R}} \subset \mathcal{L}_{\text{RE}}$.
- Wir wählen eine Sprache L', so dass $L' \notin \mathcal{L}_{\text{R}}$ und beweisen $L' \leq_{\text{R/EE}} L$.

  Geeignete Sprachen als L' sind: $L^{\complement}_{empty}, L^{\complement}_{diag}, L_{\text{H}}, L_{\text{U}}, L_{\text{H},\lambda}$. (Alle im Buch bewiesen)
- Satz von Rice

Für den **Satz von Rice**:

- Wir können mit diesem Satz nur $L \notin \mathcal{L}_{\mathrm{R}}$ beweisen!
- Wir haben folgende Bedingungen:

  1. $L \subseteq$ KodTM
  2. $\exists$ TM M: $\mathrm{Kod}(M) \in L$
  3. $\exists$ TM M: $\mathrm{Kod}(M) \notin L$
  4. $\forall$ TM M_1, M_2: $L(M_1) = L(M_2) \implies (\mathrm{Kod}(M_1) \in L \iff \mathrm{Kod}(M_2) \in L)$

  Für den letzten Punkt (4) muss man überprüfen, ob in der Definition von $L = \{\mathrm{Kod}(M) \mid M \text{ ist TM und } ...\}$ überall nur $L(M)$ vorkommt und nirgends M direkt. Beziehungsweise reicht es, wenn man die Bedingung so umschreiben kann, dass sie nur noch durch $L(M)$ beschrieben ist.

4.3 $L \in \mathcal{L}_{\mathbf{RE}}$

Wir beschreiben eine TM M mit $L(M) = L$, die nicht immer halten muss.

Meistens muss die TM eine Eigenschaft, für alle möglichen Wörter prüfen. (Bsp: $\mathrm{Kod}(M_1) \in L_{\mathrm{H}}^{\complement}$: Wir gehen alle Wörter durch, um dasjenige zu finden, für das M_1 hält.)

Wir verwenden oft einen von den folgenden 2 Tricks, um dies zu tun:

- Da es für jede NTM M', eine TM M gibt, so dass $L(M') = L(M)$, können wir eine solche definieren, für die $L(M') = L$ gilt.
- Die andere Variante, ist die parallele Simulation von Wörtern, bei dem man das Diagonalisierungsverfahren aus dem Buch verwendet. (Bsp: Beweis $L_{\mathrm{empty}} \in \mathcal{L}_{\mathrm{RE}}$, S. 156 Buch)

4.4 $L \notin \mathcal{L}_{\mathbf{RE}}$

Hier haben wir 2 mögliche (offizielle) Methoden:

- Diagonalisierungsargument mit Widerspruch, wie beim Beweis von $L_{\mathrm{diag}} \notin \mathcal{L}_{\mathrm{RE}}$.
- Widerspruchsbeweis mit der Aussage $L \in \mathcal{L}_{\mathrm{RE}} \wedge L^{\complement} \in \mathcal{L}_{\mathrm{RE}} \implies L \in \mathcal{L}_{\mathrm{R}}$.

Inoffiziell könnten wir auch die EE-Reduktion verwenden, wird aber weder in der Vorlesung noch im Buch erwähnt.

4.5 EE- und R-Reduktionen: Tipps und Tricks

- Die vorgeschaltete TM A muss immer terminieren! I.e. sie muss ein Algorithmus sein.
- Die Eingabe sollte immer zuerst auf die Richtige Form überprüft werden!

  Auch im Korrektsheitsbeweis, sollte dieser Fall als erstes abgehandelt werden.

- Für Korrektheit müssen wir immer $x \in L_1 \iff A(x) \in L_2$ beweisen.
- Wir verwenden meistens folgende 2 Tricks:
  1. Transitionen nach q_{accept} oder q_{reject} umleiten nach q_{reject}/q_{accept} oder einer **Endlosschleife**.
  2. TM M' konstruieren, die ihre Eingabe ignoriert und immer dasselbe tut (z.B. eine TM dessen Kodierung gegeben ist, auf ein fixes Wort simuliern).
- Die Kodierung einer TM generieren, dessen Sprache gewisse Eigenschaften hat(z.B. sie akzeptiert alle Eingaben, läuft immer unendlich etc.)

5 Polynomialzeitreduktionen

Typische Aufgabe: L ist NP-Vollständig. Dann müssen wir (i) L in NP und (ii) L ist NP-schwer zeigen.

(i) Wir beschreiben eine NTM M, so dass $L(M) = L$. M errät (nichtdeterministisch) ein Zertifikat und verfiziert dies (deterministisch) in Polynomialzeit. M akzeptiert, wenn die Verfikation erfolgreich ist.

M akzeptiert $\iff$ M hat eine akzeptierende Berechnung

(ii)
- Wir nehmen eine Sprache L' die NP-Schwer ist und zeigen $L' \leq_p L$.
  Beweisidee:
  Wir zeigen eine Reduktion indem wir einen Polynomialzeit Algorithmus A beschreiben, so dass $x \in L \iff A(x) \in L'$. Wir müssen also folgende 2 Punkte für A beweisen:
  - $x \in L \iff A(x) \in L'$ (meist recht komplex, beide Richtungen einzeln beweisen)
  - A läuft in Polynomialzeit (meist trivial, es reicht eine High-Level Begründung zu geben)
- Wir könnten es auch direkt beweisen(wie Beweis vom Satz von Cook). Dies ist aber meist zu komplex.

6 Grammatiken

Beispiel 10.6

Sei $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$

Beweis durch Widerspruch:

Sei L kontextfrei. Dann gilt das Pumping Lemma für kontextfreie Sprachen.

Sei n_L die Konstante aus dem Pumping Lemma.

Dann wählen wir $z = a^{n_L} b^{n_L} c^{n_L}, |z| \geq n_L, z \in L$.

Dann gilt für jede Partition $z = uvwxy$ mit (i) $|vx| \geq 1$ und (ii) $|vwx| \leq n_L$, auch (iii) $\{uv^i wx^i y \mid i \in \mathbb{N}\}$.